Final Project Documentation

SE 319: Construction of User Interfaces

12/11/2023

Chris Smith | clsmith3@iastate.edu

Xuan Wen Loo | xuanwen@iastate.edu

Professor Abraham Aldaco

INDEX

[**Project Description**](#_nbvfp5aenvsa) **3**

[**Software Functionality Diagram**](#_412m1ev70ddc) **3**

[**Files and Directory Architecture**](#_iqh9v0hcbsnq) **4**

[Description and diagram of the files and directory architecture](#_iyp1ffkp2t8n) 4

[Description and diagram of client - server architecture](#_hlhujv9sciye) 5

[Description and diagram of the logical architecture](#_e9b6vf5d12vr) 6

[Database and API diagrams](#_qufim4ez3uwj) 7

[Explanation of each web view](#_8sk4z04rhwzo) 8

[**Installation manual**](#_8pwz8j27e45b) **10**

[**Codes**](#_nup5s9o0eaat) **12**

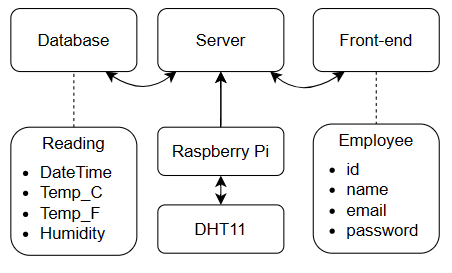
### Project Description

This project is to create an app of a temperature and humidity tracker for a company, Wan Yeen Tradings. The program enables a client to remotely view the temperature and humidity of wherever the sensor is set up. The DHT11 temperature and humidity sensor is connected to a raspberry pi which runs a program to record the data and regularly make post requests to the database to save it. A Node server handles the post requests from the raspberry pi as well as all other requests coming from the REACT application.

The REACT application requires a user to log in with an email and password before accessing the core content. Once a user enters the main page, the most recent one hundred readings from the database are immediately organized and displayed for the user. This page also displays the number of readings stored in the database and gives the user the option to clear all but the most recent 100 readings.

The ‘Company’ view allows the user to see information about the client, Wan Yeen Tradings, including a company description and contact information. The ‘Developers’ view displays information about the course and the wonderful team that developed this software.

### Software Functionality Diagram



The server is connected to Raspberry Pi and DHT 11 sensor that obtains temperature and humidity readings. The data obtained is stored in the MongoDB database that could be retrieved by the Node Server and displayed in the front end.

### Files and Directory Architecture

#### Description and diagram of the files and directory architecture

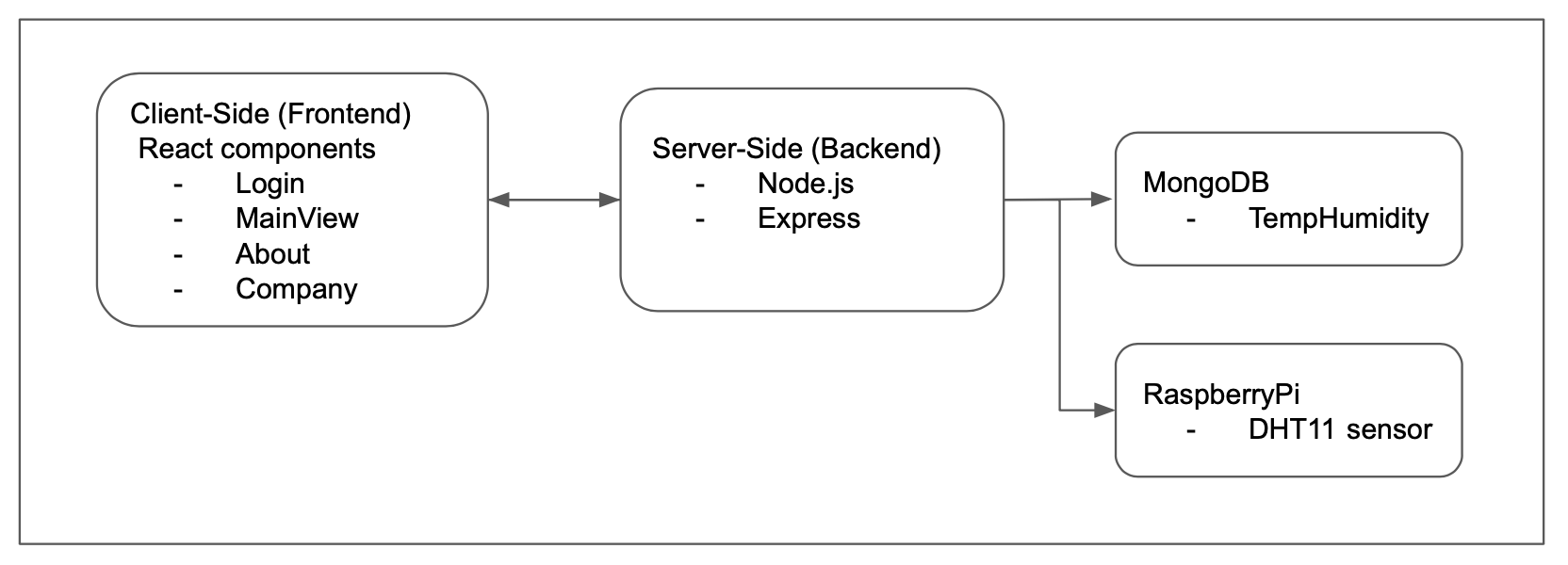
1. src Directory:

* Components Directory (Webpages views):
  + About.js: React component for the ‘Developers’ view. Show information of the course and students
  + Company.js: React component for the ‘Company’ view. Show information of the company (client)
  + Login.js: React component for user authentication.
  + MainView.js: React component for the main page displaying temperature and humidity readings.
* App\_Mongo.js: Entry point for the React application with Mongo DB integration.
* App.js: Entry point for the React application/
* employees.json: JSON file containing employee data used for login.
* index.css: Styling for the React application.
* index.js: Entry point for React application rendering.

2. Root Directory:

* package-lock.json: Auto-generated file for npm package versions.
* package.json: Configuration file for npm packages. Added express, cors, body-parser, mongodb, bootstrap, etc.
* tempAndHumidityRead.py: Python script for reading temperature and humidity data.

#### Description and diagram of client - server architecture



1. Client-Side: React app (Frontend)

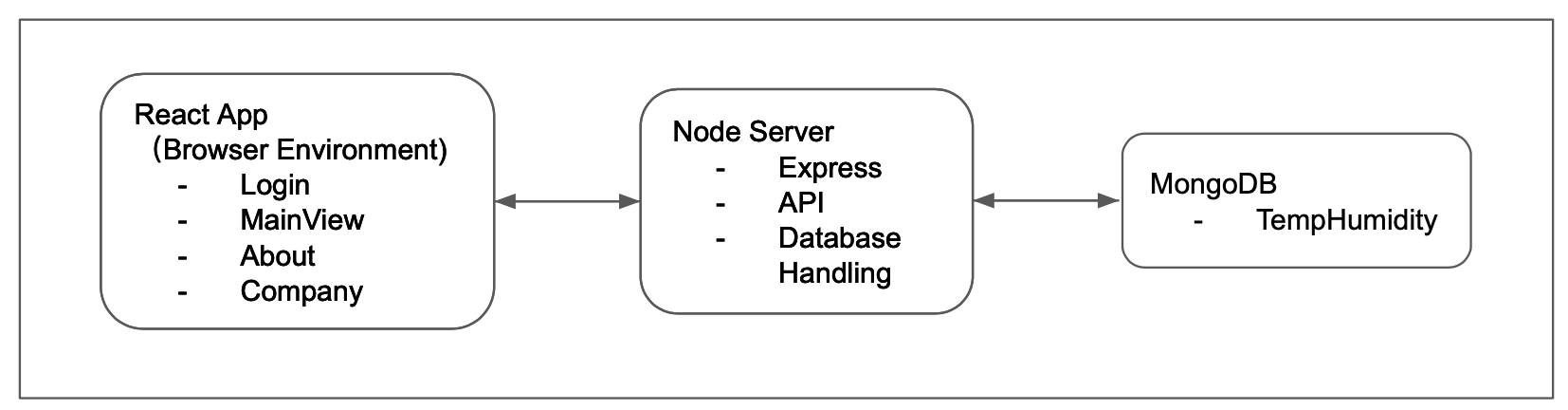
* There are 4 different react components representing different views of the application.
* The React components communicate with the server through HTTP requests (CRUD)
* User authentication is managed through the Login.js component.
* Data display: MainView.js displays temperature and humidity readings from the server.

1. Server-Side: Node.js and Express (Backend)

* The server handles API requests from the React application, as well as requests from the Raspberry Pi sensor.
* The server interacts with the MongoDB database to retrieve and store relevant data.

React components make HTTP requests to the server for data retrieval and storage. The server communicates with the MongoDB database to fetch and store temperature and humidity data. The server responds to requests from the Raspberry Pi which sends sensor readings to be stored in the database. Lastly, the React components receive responses from the server and update the UI (MainView.js) accordingly.

#### Description and diagram of the logical architecture



The diagram provides a high-level overview of the relationships and interactions between the React app, Node server, and MongoDB. Each component has its defined responsibilities, contributing to the overall functionality of the temperature and humidity tracker system.

1. React App (Frontend browser)

* Login.js: Manages user authentication through login credentials. Sends login requests to the server for verification.
* MainView.js: Responsible for displaying the main page with temperature and humidity readings. Communicates with the server to fetch and display data.
* About.js: Display information about the developers and course.
* Company.js: Display information of the client company.

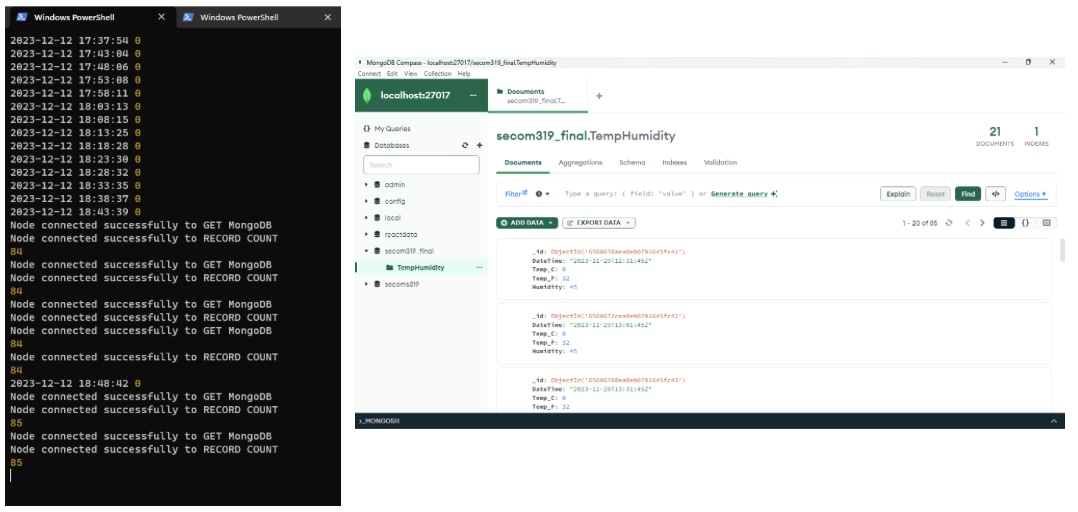
1. Node Server

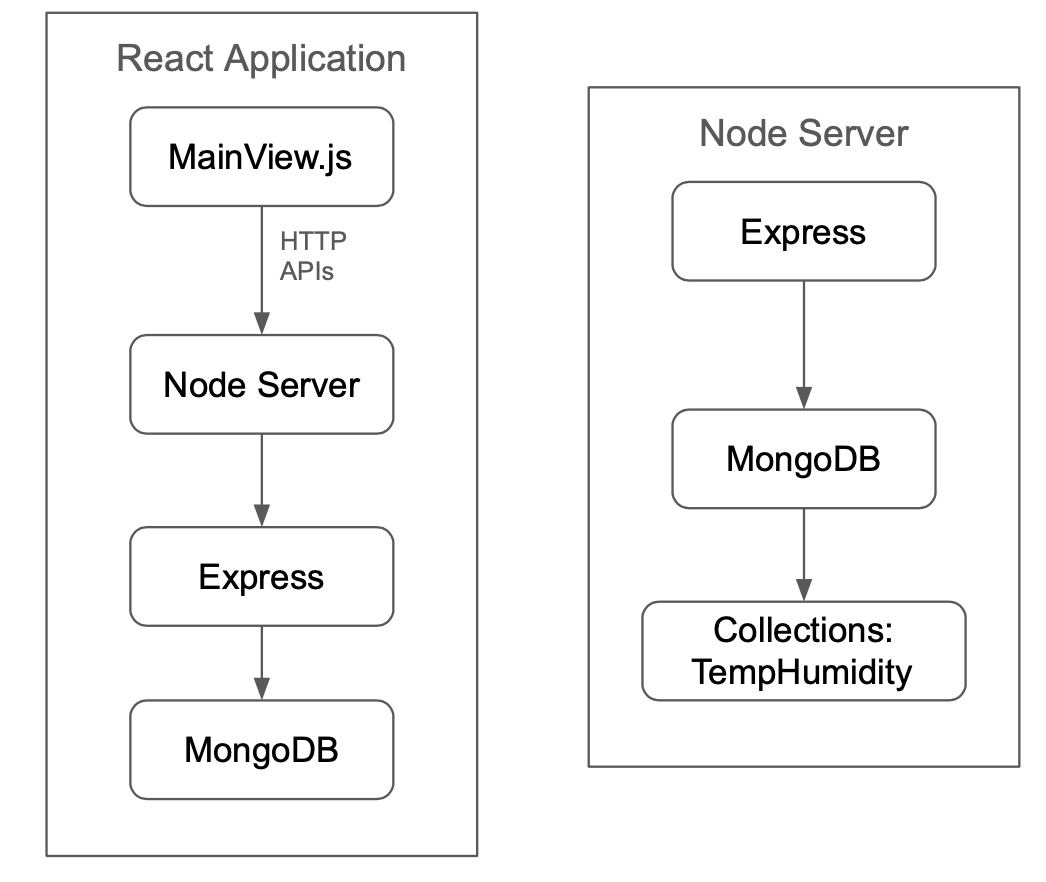
* Express/ API Handling: Defines and handles APIs for communication with the React app. Manages routes for fetching and storing data.
* Database Handling: Interacts with the MongoDB database to retrieve and store temperature and humidity data.

1. MongoDB

* Collections (TempHumidity): Provides a structured storage system for the temperature and humidity readings obtained from the sensor.

#### Database and API diagrams

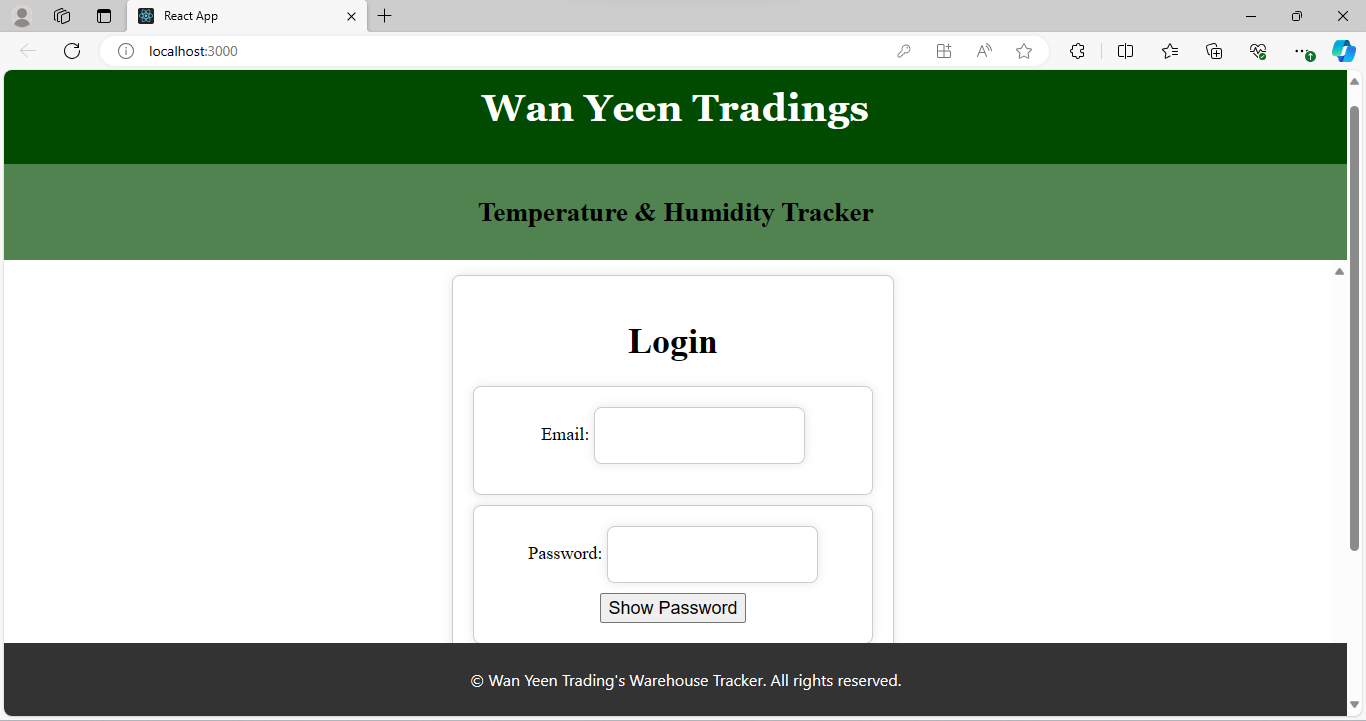




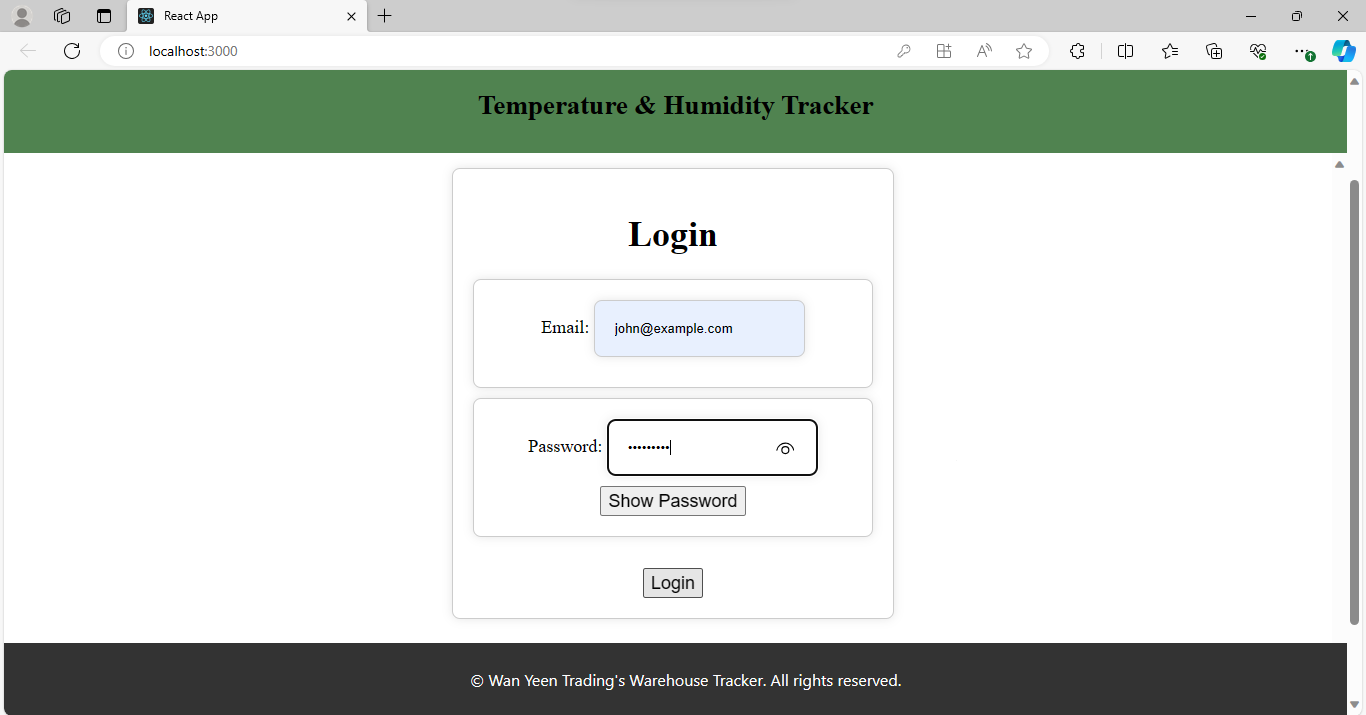
For the flow of API requests within the system, the React App, specifically MainView.js, makes HTTP requests to the Node Server that was implemented with Express to handle the API requests. The Express server interacts with MongoDB to store and fetch data.

For the flow of data within the database, the Express server, as part of the Node Server, interacts with MongoDB which contains a collection, TempHumidity, where the data from the sensor are stored in a 30-minute interval.

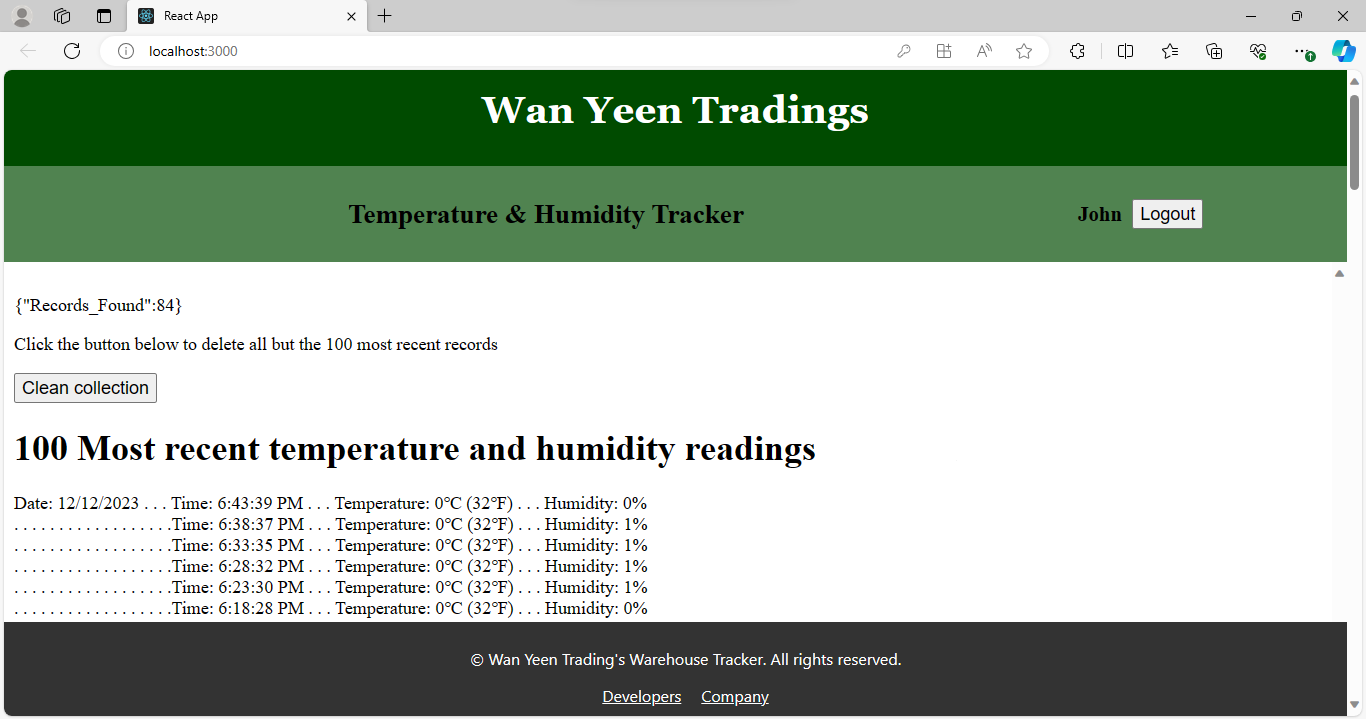
#### Explanation of each web view



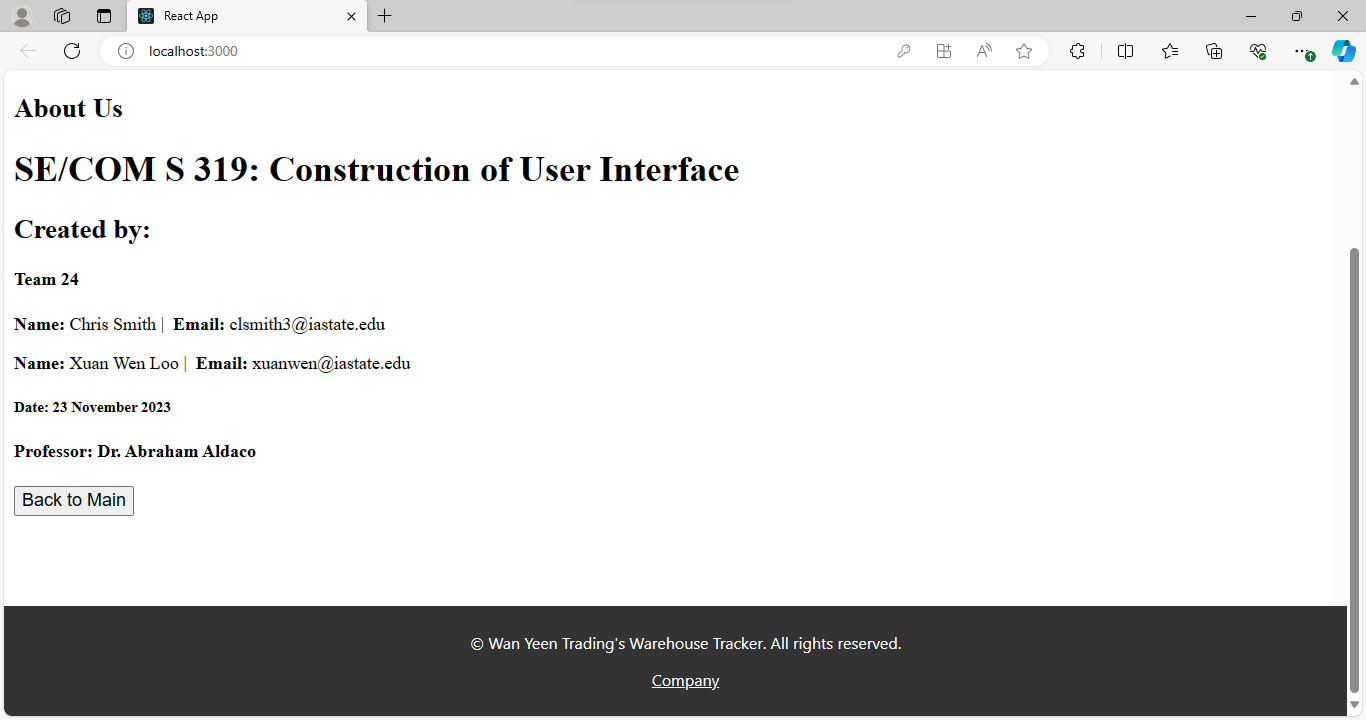
After starting the react app, the initial view will be a login page that asks for the user to input email and password.



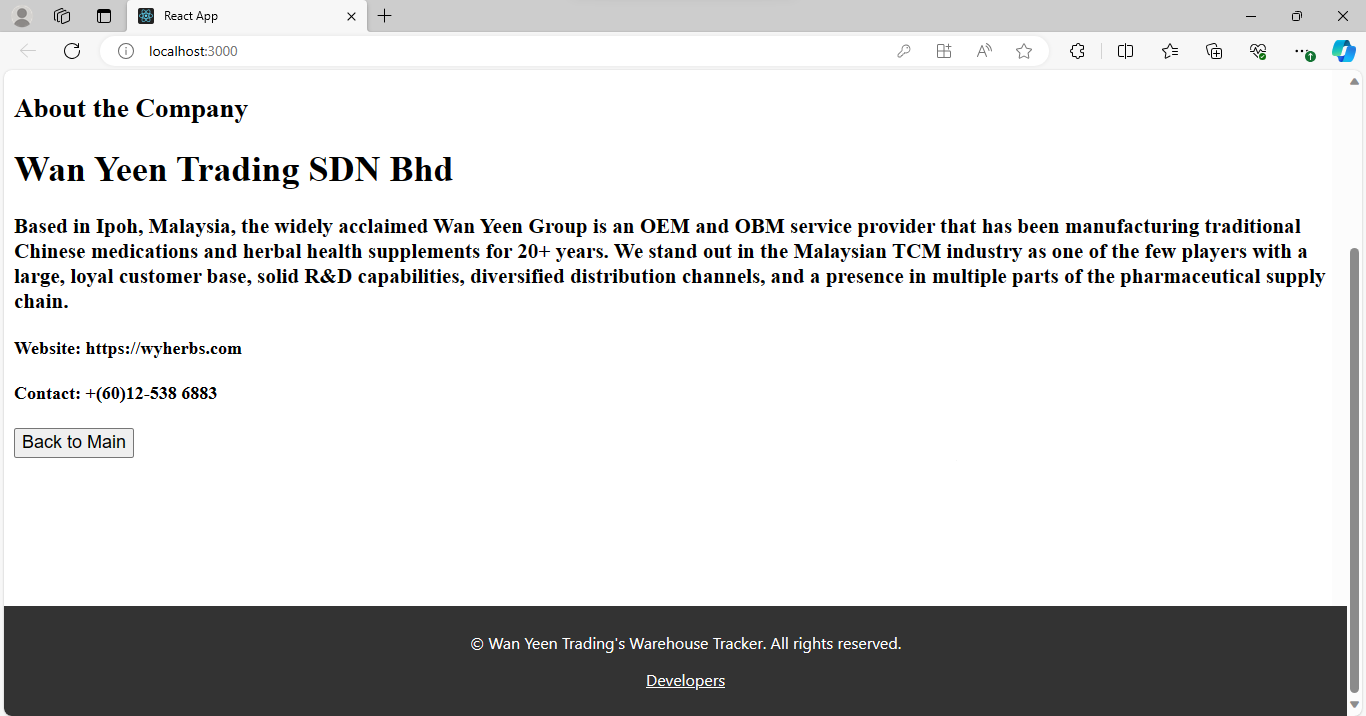
The ‘Show Password’ button will reveal the password entered on click. Upon clicking the ‘Login’ button, it will check if the email-password combination matches the data in “employees.json”. If yes, the user will be directed to the Main View. Otherwise, it will show an error message asking the user to re-enter their login credentials.



This will be the view that shows up upon successful login. It shows the 100 most recent records of temperature and humidity. There is a ‘Clean collection’ button that implements a CRUD: Delete that clears the data on click. From the MainView, users can navigate to ‘Developers’ and ‘Company’. There’s a ‘Logout’ button at the subheader that directs the user back to the login page.



This is the ‘Developers’ view that shows the students information and course detail. The ‘Back to Main’ button will direct the user back to the MainView on click. Users can navigate to ‘Company’ or Logout from this view.



This is the ‘Company’ view that shows the information of the company this project is developed for. Similarly, the ‘Back to Main’ button will direct the user back to the MainView on click. Users can navigate to ‘Developers’ or Logout from this view.

### Installation manual

Backend instructions:  
 For the sensor you need a raspberry pi connected to a DHT11 temperature and humidity sensor. To install the sensor connect the power(+) wire to pin-2, the ground(-) wire to pin-6, and the data wire to pin-7 which is GPIO4. Other wire setups can work depending on the configuration of your pi’s pins, but changing where the data wire is connected requires a small adjustment to the ‘temperatureAndHumidityRead.py’ code. This adjustment can be made by modifying the line “dhtDevice = Adafruit\_dht.DHT11(board.D4)” and changing “4” to the number of the GPIO port you wish to use. Note that the number of the GPIO does not necessarily match the number of the pin.

To run the ‘tempuratureAndHumidityRead.py’ program you will need to download certain supporting software to the device. The three key libraries needed are Adafruit-blinka, Adafruit-CircuitPython-BusDevice, and Adafruit-circuitpython-dht. Assuming your raspberry pi is using a Linux operating system with Python pre-installed, these can be downloaded by simply opening the terminal and typing “pip install <insert name of library>”. These libraries are essential as they enable the program to interface with the hardware of the DHT11 sensor. If Python/pip are not already set up on your device, you can add them from the terminal by using the commands “sudo apt-get update”, “sudo apt-get install python3”, and then “sudo apt-get install python3-pip”.

Once these libraries are successfully installed there is one last step before the program can be run. Open the ‘temperatureAndHumidityRead.py’ file and look for the line labeled with “#Server info”. In the line immediately below you must adjust the URL to reflect the IP address where you are running the application backend program ‘App\_Mongo.js’. At this point the code is ready to run by using the ‘run’ button in the editor or running it from the terminal, but you can also take this moment to make desired adjustments to the function if desired. The code is set up to be easily adjustable for key components. By default, the code is set to collect data and post it to the database once every 30 minutes, but by changing the value of ‘waitTime’ you can set it to run as often as you want by setting waitTime equal to the desired timespan in seconds. However it is important to note that running more than once per 2 seconds can cause issues with collecting accurate data.

In order for data to be added or retrieved from the database the backend server must be running. The server only requires the user device to have NodeJS installed, it can then be run with the terminal command ‘node App\_Mongo.js’. This program will handle requests made to the MongoDB database. If you wish to change the database these programs interface with then you must change the variable ‘dbName’ to reflect the name of the new database.

### 

### Codes

**src**

App\_Mongo.js

var express = require("express");

var cors = require("cors");

var app = express();

var fs = require("fs");

var bodyParser = require("body-parser");

const { MongoClient } = require("mongodb");

// Mongo

const url = "mongodb://127.0.0.1:27017";

const dbName = "secom319\_final";

const client = new MongoClient(url);

const db = client.db(dbName);

app.use(bodyParser.json());

app.use(cors());

app.get("/listTemps", async (req, res) => {

    await client.connect();

    console.log("Node connected successfully to GET MongoDB");

    const query = {};

    const results = await db

        .collection("TempHumidity")

        .find(query)

        .sort({ DateTime: -1})

        .limit(100)

        .toArray();

    //console.log(results);

    res.status(200);

    res.send(results);

});

app.get("/recordCount", async (req, res) => {

    try{

    await client.connect();

    console.log("Node connected successfully to RECORD COUNT");

    const itemCount = await db

        .collection("TempHumidity")

        .countDocuments();

    const itemArray = {"Records\_Found" : itemCount};

    console.log(itemCount);

    //res.sendStatus(200);

    res.send(itemArray);

    } catch (error) {

        console.error('Error fetching record count:', error.message);

        res.status(500).send('Internal Server Error');

    } finally {

        //await client.close(); // Close the MongoDB connection

    }

});

app.post("/postTemp", async (req, res) => {

    await client.connect();

    const keys = Object.keys(req.body);

    const values = Object.values(req.body);

    const dateTime = values[0];

    const tempC = values[1];

    const tempF = values[2];

    const humidity = values[3];

    console.log(dateTime, tempC);

    const newDocument = {

        "DateTime" : dateTime,

        "Temp\_C" : tempC,

        "Temp\_F" : tempF,

        "Humidity" : humidity

    };

    const results = await db.collection("TempHumidity").insertOne(newDocument);

    res.status(200);

    res.send(results);

});

app.delete("/deleteRecords", async (req, res) => {

    try{

    await client.connect();

    const query = {};

    const recentItems = await db.collection("TempHumidity")

        .find({})

        .sort({\_id: -1})

        .limit(100)

        .toArray();

    if(recentItems.length >100){

        const recentItemIds = recentItems.map((item) => item.\_id);

        const deleteQuery = { \_id: { $nin: recentItemIds } };

        const deleteResult = await db.collection("TempHumidity").deleteMany(deleteQuery);

        console.log(`Deleted ${deleteResult.deletedCount} items.`);

    } else{

        console.log('Less than 101 items in the collection. No deletion performed.');

        res.send('Less than 101 items in the collection. No deletion performed.').status(200);

    }

    } catch (error){

        console.error('Error deleting records:', error);

        res.status(500).send('Internal Server Error');

    } finally{

        //if (!results) res.send("Not Found").status(404);

    }

});

const port = "8081";

const host = "localhost";

app.listen(port, () => {

    console.log("App listening at http://%s:%s", host, port);

});

App.css

.App {

  text-align: center;

}

.App-logo {

  height: 40vmin;

  pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

  .App-logo {

    animation: App-logo-spin infinite 20s linear;

  }

}

.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-link {

  color: #61dafb;

}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

App.js

import React, { useState } from "react";

// const React = require("react");

// const {useState} = require("react");

import Login from "./components/Login";

import MainView from "./components/MainView";

import SubView from "./components/SubView";

import About from "./components/About";

import Company from "./components/Company";

function App() {

  const [currentView, setCurrentView] = useState("login");

  const [userName, setUserName] = useState("");

  const handleLoginSuccess = (name) => {

    setUserName(name);

    setCurrentView("main");

  };

  const handleMainToSubView = () => {

    setCurrentView("sub");

  };

  const handleSubViewToMain = () => {

    setCurrentView("main");

  };

  const handleLogout = () => {

    setCurrentView("login");

  };

  const navToAbout = () => {

    setCurrentView("about");

  };

  const navToCompany = () => {

    setCurrentView("company");

  };

  return (

    <div>

      {currentView === "login" && <Login onLoginSuccess={handleLoginSuccess} />}

      {currentView === "main" && (

        <MainView

          userName={userName}

          onMainToSubView={handleMainToSubView}

          onLogout={handleLogout}

          onAbout={navToAbout}

          onCompany={navToCompany}

        />

      )}

      {currentView === "sub" && (

        <SubView

          userName={userName}

          onSubViewToMain={handleSubViewToMain}

          onLogout={handleLogout}

          onAbout={navToAbout}

          onCompany={navToCompany}

        />

      )}

      {currentView === "about" && (

        <About

          userName={userName}

          onSubViewToMain={handleSubViewToMain}

          onLogout={handleLogout}

          onCompany={navToCompany}

        />

      )}

      {currentView === "company" && (

        <Company

          userName={userName}

          onSubViewToMain={handleSubViewToMain}

          onLogout={handleLogout}

          onAbout={navToAbout}

        />

      )}

    </div>

  );

}

export default App;

employees.json

[

  {

    "id": 1,

    "name": "John",

    "email": "john@example.com",

    "password": "password1"

  },

  {

    "id": 2,

    "name": "Mary",

    "email": "mary@example.com",

    "password": "password2"

  }

]

Index.css

body {

  margin: 0;

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

header {

  background-color: rgb(0, 75, 0);

  color: white;

  padding: 10px;

  text-align: center;

  font-family: Georgia, 'Times New Roman', Times, serif;

  font-size: larger;

}

#subheader {

  background-color: rgba(0, 75, 0, 0.686);

  color: rgb(0, 0, 0);

  padding: 10px;

  text-align: center;

  font-family:'Times New Roman', Times, serif;

  font-size: large;

}

#subheader2{

  display: grid;

  grid-template-columns: auto auto auto auto auto auto;

  background-color: rgba(0, 75, 0, 0.686);

  color: rgb(0, 0, 0);

  padding: 10px;

  text-align: center;

  font-family:'Times New Roman', Times, serif;

  font-size: large;

}

#title {

  grid-column: 1/6;

}

#content{

  padding-top: 15px;

  padding-left: 10px;

  font-family:'Times New Roman', Times, serif;

  font-size: large;

  min-height: 100vh;

  overflow-y: scroll; /\* Allow vertical scrolling \*/

}

#content2{

  padding-top: 15px;

  padding-left: 10px;

  font-family:'Times New Roman', Times, serif;

  font-size: large;

  min-height: 600vh;

  overflow-y: scroll; /\* Allow vertical scrolling \*/

}

footer {

  background-color: #333;

  color: white;

  padding: 10px;

  text-align: center;

  position: fixed;

  bottom: 0;

  width: 100%;

}

button{

  font-size: large;

  width: max-content;

  height: 30px;

}

#login {

  text-align: center;

  padding: 10px;

  margin-left: 20px;

  margin-right: 20px;

  border-color: black;

}

reportWebVitals.js

const reportWebVitals = onPerfEntry => {

  if (onPerfEntry && onPerfEntry instanceof Function) {

    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {

      getCLS(onPerfEntry);

      getFID(onPerfEntry);

      getFCP(onPerfEntry);

      getLCP(onPerfEntry);

      getTTFB(onPerfEntry);

    });

  }

};

export default reportWebVitals;

**Components**

About.js

import React from "react";

function About({userName, onSubViewToMain, onLogout, onCompany}) {

  return (

    <div>

      <header>

        <h1>Wan Yeen Tradings</h1>

      </header>

      <div id="subheader2">

        <div id="title"><h2>Temperature & Humidity Tracker</h2></div>

        <div id="username" style={{ display: "flex", alignItems: "center" }}>

          {userName && (

            <h3 style={{ display: "inline", marginRight: "10px" }}>

              {userName}

            </h3>

          )}

          <button onClick={onLogout}>Logout</button>

        </div>

      </div>

      <div id="content">

      <h2>About Us</h2>

      <h1>SE/COM S 319: Construction of User Interface</h1>

    <h2>Created by: </h2>

    <section id="creator">

        <h4 id="team">Team 24</h4>

        <div class="container-sm text-center">

            <div class="row">

                <div class="col student">

                    <p>

                        <strong>Name: </strong>Chris Smith | &nbsp;

                        <strong>Email: </strong>clsmith3@iastate.edu

                    </p>

                </div>

                <div class="col student">

                    <p>

                        <strong>Name: </strong>Xuan Wen Loo | &nbsp;

                        <strong>Email: </strong>xuanwen@iastate.edu

                    </p>

                </div>

            </div>

        </div>

    </section>

    <h5>Date: 23 November 2023</h5>

    <h4>Professor: Dr. Abraham Aldaco</h4>

      <button onClick={onSubViewToMain}>Back to Main</button>

      </div>

      <footer>

        <p>&copy; Wan Yeen Trading's Warehouse Tracker. All rights reserved.</p>

        <p onClick={onCompany}><u>Company</u></p>

      </footer>

    </div>

  );

}

export default About;

Company.js

import React from "react";

function Company({ userName, onSubViewToMain, onLogout, onAbout }) {

  return (

    <div>

      <header>

        <h1>Wan Yeen Tradings</h1>

      </header>

      <div id="subheader2">

        <div id="title">

          <h2>Temperature & Humidity Tracker</h2>

        </div>

        <div id="username" style={{ display: "flex", alignItems: "center" }}>

          {userName && (

            <h3 style={{ display: "inline", marginRight: "10px" }}>

              {userName}

            </h3>

          )}

          <button onClick={onLogout}>Logout</button>

        </div>

      </div>

      <div id="content">

        <h2>About the Company</h2>

        <h1>Wan Yeen Trading SDN Bhd</h1>

        <div>

          <h3>

            Based in Ipoh, Malaysia, the widely acclaimed Wan Yeen Group is an

            OEM and OBM service provider that has been manufacturing traditional

            Chinese medications and herbal health supplements for 20+ years. We

            stand out in the Malaysian TCM industry as one of the few players

            with a large, loyal customer base, solid R&D capabilities,

            diversified distribution channels, and a presence in multiple parts

            of the pharmaceutical supply chain.

          </h3>

        </div>

        <h4>Website: https://wyherbs.com</h4>

        <h4>Contact: +(60)12-538 6883</h4>

        <button onClick={onSubViewToMain}>Back to Main</button>

      </div>

      <footer>

        <p>&copy; Wan Yeen Trading's Warehouse Tracker. All rights reserved.</p>

        <p onClick={onAbout}><u>Developers</u></p>

      </footer>

    </div>

  );

}

export default Company;

Login.js

import React, { useState, useEffect } from "react";

import usersData from "../employees.json";

function Login({ onLoginSuccess }) {

  const [email, setEmail] = useState("");

  const [password, setPassword] = useState("");

  const [showPassword, setShowPassword] = useState(false);

  const [error, setError] = useState("");

  const handleLogin = () => {

    // Check if the email and password match any user data

    const user = usersData.find(

      (user) => user.email === email && user.password === password

    );

    if (user) {

      // Successful login

      onLoginSuccess(user.name);

    } else {

      // Failed login

      setError("Invalid email or password");

    }

  };

  // Styling

  const styles = {

    loginContainer: {

      maxWidth: "400px",

      margin: "auto",

      marginBottom: "10px",

      padding: "20px",

      border: "1px solid #ccc",

      borderRadius: "8px",

      boxShadow: "0 0 10px rgba(0, 0, 0, 0.1)",

      backgroundColor: "#fff",

    },

    inputContainer: {

      marginBottom: "10px",

      width: "70%",

    },

    inputField: {

      width: "100%",

      padding: "8px",

      boxSizing: "border-box",

      fontSize: "21px",

    },

    loginButton: {

      backgroundColor: "#4caf50",

      color: "#fff",

      padding: "15px",

      border: "none",

      borderRadius: "5px",

      cursor: "pointer",

      width: "100%",

    },

    error: {

      marginTop: "10px",

    },

  };

  return (

    <div>

      <header>

        <h1>Wan Yeen Tradings</h1>

      </header>

      <div id="subheader">

        <h2>Temperature & Humidity Tracker</h2>

      </div>

<div id="content">

      <div id="login" style={styles.loginContainer}>

        <h1>Login</h1>

        <div style={styles.loginContainer}>

          <label>Email: </label>

          <input

            type="email"

            value={email}

            onChange={(e) => setEmail(e.target.value)}

            style={styles.loginContainer}

          />

        </div>

        <div style={styles.loginContainer}>

          <label>Password: </label>

          <input

            type={showPassword ? "text" : "password"}

            value={password}

            onChange={(e) => setPassword(e.target.value)}

            style={styles.loginContainer}

          />

          <button onClick={() => setShowPassword(!showPassword)}>

            {showPassword ? "Hide" : "Show"} Password

          </button>

        </div>

        <div>&nbsp;</div>

        <button onClick={handleLogin}>Login</button>

        {error && <p style={{ color: "red", ...styles.error }}>{error}</p>}

      </div>

      </div>

      <footer>

        <p>&copy; Wan Yeen Trading's Warehouse Tracker. All rights reserved.</p>

      </footer>

    </div>

  );

}

export default Login;

MainView.js

import React, { useState, useEffect } from "react";

function MainView({ userName, onLogout, onAbout, onCompany }) {

  const [data, setData] = useState([]);

  let result ="";

  let dateStore ="";

  const [itemCount, setCount]= useState([]);

  useEffect(() => {

    // Fetch data when the component mounts

    fetchData();

    recordCount();

  }, []);

  async function fetchData() {

    try {

      const response = await fetch('http://localhost:8081/listTemps');

      // response.json().then(data=>{

      //   setData(data);

      // });

      if (!response.ok) {

        throw new Error('Failed to fetch data');

      }

      const data = await response.json();

      setData(data);

      //result = await response; // Assuming the response contains JSON data

    } catch (error) {

      console.error('Error fetching data:', error.message);

    }

  }

  async function recordCount() {

    //console.log("count");

    try {

      const response = await fetch('http://localhost:8081/recordCount');

      if(!response.ok){

        throw new Error('Failed to fetch itemCount');

      }

      const itemCount = await response.text();

      setCount(itemCount);

    } catch (error) {

      console.error('Error fetching data:', error.message);

    }

  }

  async function handleDelete(){

    try{

      const response = await fetch('http://localhost:8081/deleteRecords', {

        method: 'DELETE'

      });

      if (response.ok) {

        console.log('Request processed.');

      } else {

        console.error('Error deleting records:', response.statusText);

      }

    } catch (error) {

      console.error('Error:', error.message);

    }

  };

  return (

    <div>

      <header>

        <h1>Wan Yeen Tradings</h1>

      </header>

      <div id="subheader2">

        <div id="title">

          <h2>Temperature & Humidity Tracker</h2>

        </div>

        <div id="username" style={{ display: "flex", alignItems: "center" }}>

          {userName && (

            <h3 style={{ display: "inline", marginRight: "10px" }}>

              {userName}

            </h3>

          )}

          <button onClick={onLogout}>Logout</button>

        </div>

      </div>

      <div id="content2">

        <div>

          <p>{itemCount}</p>

          <p>Click the button below to delete all but the 100 most recent records </p>

            <button onClick={handleDelete}>Clean collection</button>

        </div>

        <h1>100 Most recent temperature and humidity readings</h1>

        {data.map(item=>(

          <div key={item.\_id}>{(() =>{

            const dateTime = new Date(item.DateTime);

            const spacing = ' . . . ';

            const dateSpace = ' . . . . . . . . . . . . . . . . . .';

            if(dateStore !== dateTime.toLocaleDateString()){

              dateStore = dateTime.toLocaleDateString();

              result = `Date: ${dateTime.toLocaleDateString()}${spacing}Time: ${dateTime.toLocaleTimeString()}${spacing}Temperature: ${item.Temp\_C}℃ (${item.Temp\_F}℉)${spacing}Humidity: ${item.Humidity}%`;

            }else{

              result = `${dateSpace}Time: ${dateTime.toLocaleTimeString()}${spacing}Temperature: ${item.Temp\_C}℃ (${item.Temp\_F}℉)${spacing}Humidity: ${item.Humidity}%`;

            }

            return result;

           })()}

           </div>

        ))}

      </div>

      <footer>

        <p>&copy; Wan Yeen Trading's Warehouse Tracker. All rights reserved.</p>

        <div>

          <p

            style={{

              display: "inline",

              marginRight: "20px",

              cursor: "pointer",

            }}

            onClick={onAbout}

          >

            <u>Developers</u>

          </p>

          <p

            style={{ display: "inline", cursor: "pointer" }}

            onClick={onCompany}

          >

            <u>Company</u>

          </p>

        </div>

      </footer>

    </div>

  );

}

export default MainView;